(16)

AIRMICS

Army Institute for Research in Management Information and Computer Science

313 Calculator Bldg.
GA Institute of Technology
Atlanta, GA 30332

Technical Report

RESEARCH IN FUNCTIONALLY DISTRIBUTED COMPUTER SYSTEMS DEVELOPMENT

Kansas State University

Virgil Wallentine

Principal Investigator

Approved for public release; distribution unlimited

VOLUME XIX

ROLL–BACK AND RECOVERY IN DISTRIBUTED DATA BASE SYSTEMS

U.S. ARMY COMPUTER SYSTEMS COMMAND FT BELVOIR, VA 22060

(9) Interim rept.,

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. ~~REPORT NUMBER~~ (6) Research in Functionally Distributed Computer Systems Development. Volume XIV.	2. GOVT ACCESSION NO. AD-A103102	3. RECIPIENT'S CATALOG NUMBER
4. ROLL-BACK AND RECOVERY IN DISTRIBUTED DATA BASE SYSTEMS.		5. ~~TYPE OF REPORT & PERIOD COVERED~~ Iterim
		6. PERFORMING ORG. REPORT NUMBER (14) CS-77-5
7. AUTHOR(s) (10) Paul S./Fisher J. Fred/Maryanski		8. ~~CONTRACT OR~~ GRANT NUMBER(s) (15) DAAG 29-76-G-0108
9. PERFORMING ORGANIZATION NAME AND ADDRESS Kansas State University Department of Computer Science Manhattan, KS 66506		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS (11)
11. CONTROLLING OFFICE NAME AND ADDRESS US Army Research Office P O Box 12211 Research Triangle Park, NC 27700		12. ~~REPORT DATE~~ Feb~~ruary 1~~977
		13. NUMBER OF PAGES 19 pages
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) US Army Computer Systems Command Attn: CSCS-AT Ft. Belvoir, VA 22060 (12) 24		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

The findings in this report are not to be construed as an official Department of the Army position, unless so designated by other authorized documents.

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

DDBMS

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

-over-

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

Unclassified
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

391123 [illegible]

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE(When Data Entered)

-ABSTRACT-

One of the major obstacles to the widespread development and utilization of distributed data base management systems is the lack of an efficient recovery technique. A methodology is presented here for recovery of distributed data bases. The central operation of the recovery technique is rollback of a data base application task on the processor which controls access to the data. The rollback procedure restores the data base to its original state prior to the execution of the application task and determines the set of applications tasks which may have been effected by that task. Tasks that have not operated upon data altered by tasks being rolled back are not affected by the procedure. The rollback procedure attempts to minimize the the time and space requirements for recovery.

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE(When Data Entered)

e

ROLLBACK AND RECOVERY IN DISTRIBUTED DATA BASE MANAGEMENT SYSTEMS[1]

Technical Report CS 77-05

Fred J. Maryanski

Paul S. Fisher

Computer Science Department

Kansas State University

Manhattan, Kansas

66506

February, 1977

[1]The work reported herein was supported by the United States Army Computer Systems Command, Grant No. DAAG 29-76-G-0108.

e

Abstract

One of the major obstacles to the widespread develop‗nt and utilization of distributed data base management systems is the lack of an efficient recovery technique. A methodology is present here for recovery of distributed data bases. The central operation of the recovery technique is rollback of a data base application task on the processor which controls access to the data. The rollback procedure restores the data base to its original state prior to the execution of the application task and determines the set of application tasks which may have been effected by that task. Tasks that have not operated upon data altered by task being rolled back are not affected by the procedure. The rollback procedure attempts to minimize the time and space requirements for recovery.

1. Introduction

Distributed data base management systems (DDBMS) have the potential to make a significant impact on the way data is viewed and processed. The ability to easily and safely access data controlled by several different computers has been a long standing goal of workers in the data base area. Progress toward this goal has accelerated rapidly in both academic and industrial environments. However, many problems still require efficient solutions. One of these problem areas that can have a significant performance impact is rollback and recovery, which is the topic of this paper.

1.1 Distributed Data Base Management Systems

Many forms of distributed data base management systems have been studied [1-7]. All distributed data base configurations can be described as a collection of host, back-end, or bi-functional machines. A host machine is a processor which executes data base application tasks. The data base operations requested by a host machine are then performed by a back-end machine. A bi-functional machine contains software to carry out both the host and back-end functions. Figure 1 illustrates a DDBMS with host, back-end, and bi-functional processors.

1.2 Recovery in a DBMS

The recovery mechanism is essentially the same in all single machine data base management systems. During the execution of the data base application tasks, all data base requests are logged on a journal file (usually a tape). Whenever a data base request results in a modification to the data base, the effected pages are written onto the journal file in both before and/or after images. If a point of inactivity is reached in the processing of the data base, a checkpoint dump of DBMS' primary memory is taken. All journal and checkpoint

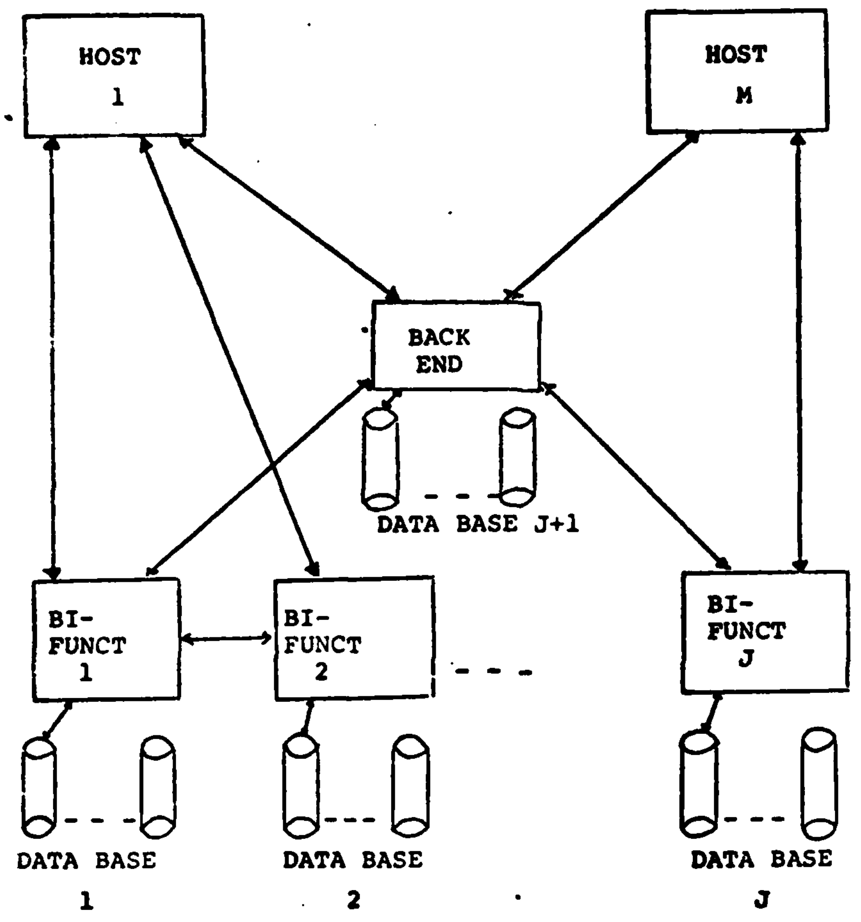


Figure 1

Distributed DBMS

information must be time stamped to insure proper sequencing of the recovery operation, which is implied by the relative position of data in the single machine case.

If an application task should terminate abnormally, its effect on the data base must be removed. It is possible in an interactive environment that data written by the terminating application task may have been used by other tasks which in turn performed additional modifications to the data base. This phenomenon of an application task using incorrect data to produce additional incorrect data can have a cascading effect throughout the data base with a large number of application tasks potentially operating with invalid information. Currently, most DBMS users avoid this problem by doing updates in a batch mode at a specified time and allowing only interactive query.

In order to undo the effect of an abnormally terminating task on the data base, a rollback procedure is necessary. The rollback procedure restores the data base to its state prior to the initiation of the erroneous application task. The precise form of a rollback procedure varies with the organization and environment of the DBMS.

In the simplest case if the run time unit of the DBMS is single threaded and operates in a batch environment, then the data base can be restored from the checkpoint taken prior to the initiation of the task.

In the more complex case if the run time unit of the DBMS is multi-threaded and operates both on-line update and batch tasks simultaneously, then one of two basic approaches is possible. The simpler approach is to roll back all tasks which either have executed or are executing to the time of initiation of the faulty task. If this procedure is followed, the integrity of all portions of the data base will be preserved, but at a potentially high cost, since it is possible that tasks which have no interaction with the terminating task may be rolled back needlessly. The more difficult approach suggests that only processes which themselves have used contaminated data be rolled back. In most cases this would involve both fewer tasks and less data. In either case, the roll-back procedure has the following basic steps:

1. The data base is restored to the first checkpoint after the initiation of the faulty task (if a checkpoint has been taken in that time frame).

2. The page images prior to each operation from the checkpoint to task initiation are written back to the data base. It should be noted that the page images are applied in reverse order (hence, the term rollback).

Either approach to the rollback of an incorrect application program will maintain data base integrity at the cost of halting some or all data base activity. While the integrity of the data base is preserved, there can be no guarantee that any erroneous information obtained by a user from an application task that terminated between the initiation of the faulty application task and the beginning of the rollback procedure can be corrected.

2. The Problem of Recovery and the DDBMS

The distribution of a data base system over several processors increases the complexity of the recovery problem. Just the interprocessor communications overhead can result in more time-consuming rollback operations if several processors are required to participate. It is also likely to be the case that within a DDBMS there is a larger number of programs interacting than in a single machine system. Hence, the complexity and effect of any rollback procedure may be compounded.

2.1 Possible DDBMS Recovery Alternatives

There are three basic approaches to recovery in a DDBMS.

1. Design the data base to permit only controlled or restricted interaction among application tasks.

2. Extend the single machine recovery mechanism to the distributed system. This approach would entail rolling back all data bases on all back-end processors in the system to the point of initiation of the faulty task.

3. Use a selective recovery mechanism to roll back only those tasks which have used data provided by the erroneous or terminated task.

2.2 Analysis of Alternatives

The three approaches to recovery in a distributed DBMS that are listed above represent widely differing philosophies. Each approach has merit for specific types of organizations and data processing facilities. The criteria used here to evaluate these alternatives are based upon the recovery procedure's effects upon system throughput, as well as data availability and the degree of data integrity maintained within the system.

The first approach, avoiding integration in the data base, has little appeal to the designer of a DDBMS, although in practice this might be the most commonly used technique since many users of data base systems on single machines feel that avoiding integration is the only reliable means of insuring data integrity. If this user philosophy were applied to a DDBMS, very inefficient utilization of the distributed system would result. In order to avoid the need for a sophisticated recovery mechanism in a DDBMS, two tasks would not be permitted to have simultaneous update capabilities to the same data base. For an application system under this requirement, the DDBMS recovery problem can be simplified. An on-line inquiry batch update system would fall in this category.

If the single machine approach is extended to a DDBMS, then the entire data base would be unavailable during recovery. In a system with a large number of back-end or bi-functional processors, this approach could result in the recovery of a small portion of a data base preventing usage of a large, correct segment of the data. The communication process involved with this technique is that a message must be transmitted to all back-end processors indicating that recovery must begin and the time of initiation of the faulty program. This system-wide recovery approach insures that all effects of the erroneous application program have been removed from the data base. The main drawbacks to this method are that access to uneffected portions of the data base is prevented and that unnecessary rollbacks may occur.

A selective recovery mechanism would overcome the main deficiency of the system-wide recovery strategy by rolling back only those application tasks that are operating with tainted data. Overall system throughput would increase under these circumstances, as would accessibility to the data base.

In order for a selective recovery scheme to be worthwhile, data base integrity must be maintained. Therefore, the scheme must be certain of including all tasks that have used incorrect data in the recovery process. In order to accomplish selective rollback, information on the interaction between application tasks must be maintained. This interaction information would take the form of a potential shared data list which can be computed from the sub-schemas of the application tasks prior to execution.

The communication overhead which potentially has the most significant performance effect in a DDBMS must not exceed one transmission to each back-end processor if the method is to be effective. The computational overhead involved in a selective recovery strategy must be maintained at a level where it is not significant in terms of system performance. A selective recovery mechanism which satisfies the performance requirements listed above has the potential for better performance in a DDBMS than the recovery schemes discussed previously.

In distributed data base systems which are highly integrated and support multi-threaded updating, a selective recovery technique is required if both performance and integrity are to be preserved.

3. A Selective Recovery Methodology

The following sections explain a procedure which meets the necessary criteria for selective rollback as described above. In addition, some comments are made concerning the role of the DBA with respect to a DDBMS system. The procedure is oriented toward a CODASYL-type DBMS, although the same general techniques are applicable to any type of data base system.

3.1 Definition-Time Recovery Processing

As previously mentioned, the proposed selective recovery methodology for distributed data base systems requires processing at data definition time, as well as run time. When a new sub-schema is created, a potential shared data list is computed by intersecting all sub-schemas of that schema. The potential shared data list indicates record and set types that are in common with other sub-schemas.

Since each application task invokes exactly one sub-schema during its execution, the potential data overlap of any two application tasks can be determined from their potential shared data lists. In order to maintain the data overlap information at execution time, the activation of a data base application task must result in a message being transmitted to all applications that have the potential to share data with that task. The message indicates the application task and sub-schema names. The information relating active application tasks and sub-schemas is maintained in the data dictionary. When an application task terminates, a similar message is sent to all tasks with intersecting potential shared data lists.

Situations may arise in integrated data bases in which application tasks share record types, but do not operate upon the contents of all data items in the record. A similar possibility is that the application task may access some data items in a read-and-print mode. In either of these situations, an incorrect value in a data item may not be critical to the function of the program. Rolling back a task due to an incorrect value in a non-critical data item would have an adverse effect on system performance.

The identity of non-critical data items is heavily application task dependent and can in no way be inferred from a sub-schema description. Since the CODASYL specifications [8] do not provide for the identifications of non-critical data items for recovery purposes, some additional mechanism for their identifications must be provided to the data base administrator. The simplest approach would be to maintain in the data dictionary a list of non-critical data items for each application task.

When sub-schemas are intersected to form the potential shared record list, non-critical data items should be removed from the intersection of the records. Only those records which intersect on critical data items should be included in the potential shared data list. Figure 2 illustrates the potential shared data list for a sample application task, A.

3.2 Logging

The functional back-end processor controls data base access and, therefore, is the appropriate location to maintain the logs of data base activity. Since a back-end processor (or a bi-functional machine serving as a back-end) may serve a large number of application tasks, the amount of recovery information must be minimized. Many existing single machine data base systems save the before and after images of each page that is altered as a result of a DML statement. For example, in a CODASYL DBMS, changes to set structure can affect many pages and consequently consume a large amount of file space. In order to reduce the amount of log file space required, an inverse DML rollback technique can be used. The inverse technique requires little primary memory, since for the most part existing DML functions can be used to perform the rollback

Critical Data Item List

Task	Record	Items
A	r_1	d_1 d_2 d_3 d_4
A	r_2	d_7 d_8
B	r_1	d_1 d_3
B	r_2	d_7 d_8
C	r_2	d_7 d_9 d_{10}
D	r_1	d_5 d_6
D	r_2	d_8 d_9 d_{10}

Potential Shared Data List (for Task A)

Task	Record
B	r_1
B	r_2
C	r_2
D	r_2

Figure 2

Sample Potential Shared Data List

operations. Table 1 lists the CODASYL DML verbs, the information that a back-end processor must save to roll back that verb, and the inverse actions.

Whenever a DML verb is executed by a back-end processor, a log file entry is written. Figure 3a depicts the format of a log file entry for a DML verb. Note that the entries have variable lengths. In addition to recording the DML verbs that have been executed, the log file must also contain restart information on all application tasks. Restart information identifies a stable point at which an application program can be restarted. By default, the system will write a restart entry whenever an application task is initiated. It is also desirable to permit the programmer the ability to indicate a restart point in a task. The CODASYL specifications do not provide a facility for this operation. However, UNIVAC's DMS-1100 has a similar feature in the LOG verb [9]. The format of a restart entry for the log file is shown in Figure 3b.

Figure 4 gives some sample DML commands and their resultant log file entries.

In certain cases, particularly a strict retrieval environment, the logging of read operations may produce a large number of entries on the log file that will never be applied in any rollback situation. Considerable rollback overhead could be saved if the data base administrator were provided the option of shutting off the logging facility for retrieval operations on selective tasks. This could be accomplished at task initiation time.

3.3 Rollback

When the DBMS software on a host processor determines that an application task has terminated abnormally, rollback procedures are initiated. The host processor notifies all back-end processors for this task that rollback must

DML Verb	Effect on Data Base	Recovery Information	Recovery Action
ACCEPT	None	N/A	N/A
CONNECT	Changes Pointers	Record & Set Names	DISCONNECT
DISCONNECT	Changes Pointers	Record & Set Names	CONNECT
ERASE	Removes Record and Members	Images of All Erased Records, Set Memberships	All Records
FIND	None	N/A	N/A
FINISH	None	N/A	N/A
FREE	None	N/A	N/A
GET	None	N/A	N/A
KEEP	None	N/A	N/A
MODIFY	Changes Record Contents	Old Record Image	MODIFY
ORDER	Changes Pointers	Set Pointers in Old Order	Reorder Using Old Pointers
READY	None	N/A	N/A
STORE	Changes Record Contents and Pointers	Record Name	ERASE
REMONITOR	None	N/A	N/A
USE	None	N/A	N/A

Table 1

Recovery Information and Action for DML Verbs

TIME	TASK	RECORD OR SET TYPE	OCCURRENCE ID	DML OPERATION	ROLLBACK INFORMATION

a. DML Verb

TIME	TASK	RESTART

b. Restart Entry

Figure 3

Log File Entries

Commands Received By Back-End Processors

Time	Command
t_0	Initiate A
t_1	Restart Point-C
t_2	A: STORE r_1
t_3	C: GET r_2
t_4	Initiate B
t_5	A: MODIFY r_2
t_6	A: MODIFY r_1
t_7	Initiate D
t_8	D: GET r_1
t_9	B: GET r_2
t_{10}	B: GET r_2

Log File

Task	Time	Occurrence	Operation
A	t_0	—	RESTART
C	t_1	—	RESTART
A	t_2	r_1	STORE
C	t_3	r_2	GET
B	t_4	—	RESTART
A	t_5	r_2	MODIFY
A	t_6	r_1	MODIFY
D	t_7	—	RESTART
D	t_8	r_1	GET
B	t_9	r_2	GET
B	t_{10}	r_2	GET

Figure 4

Sample Log File

occur and provides the task name, its initiation time, its potential shared data list, and a list of its areas which are open for update as parameters. The method for identifying the back-end processors for a given task is explained in Reference [10]. In the ensuing presentation, this task shall be known as the primary rollback task.

Each back-end processor that receives the rollback message must refuse to accept any DML operations accessing the areas updated by the terminating task. The following rollback procedure is then carried out.

1. Read the log file backwards to locate the initiation point of the task.

2. Read the log file forward from that point and perform the following operations, depending upon the entry on the log file:

a. If the entry is an update entry for the task that is being rolled back, make an entry in the update list which indicates the record or set occurrence whose contents have been modified. An entry in the update list consists of an ordered pair of record or set type and occurrence indentifiers. Copy this log file entry onto the rollback file.

b. If a MODIFY entry for the task being rolled back alters the contents of a record occurrence that was previously MODIFIED (this is indicated by the presence of the record occurrence on the update list), no action is taken with respect to either the update list or rollback file. This will insure that an updated record is restored only once to its earliest value in the rollback procedure.

c. If an entry for an application task other than the primary rollback task references a record or set occurrence for which an update list entry exists, and an entry for that task and record or set type exists in the potential shared data list, that task must also be rolled back. Rollback of this task must occur since the task may be operating with incorrect data. The task name and time of this entry are saved in a secondary rollback list.

d. All entries on the log file for tasks in the secondary rollback list are ignored.

e. All entries on the rollback file for tasks that do not have entries in the potential shared data list of the primary rollback task are ignored.

f. All entries for non-update DML commands that operate on record or or set occurrences not in the update list are ignored.

g. Any primary rollback task retrieval entries on the log file are ignored.

3. When the log file has been processed up to the time of termination, the restoration of the portions of the data base effected by the primary rollback task is performed by processing the rollback file in reverse and executing the rollback actions indicated in Table 1.

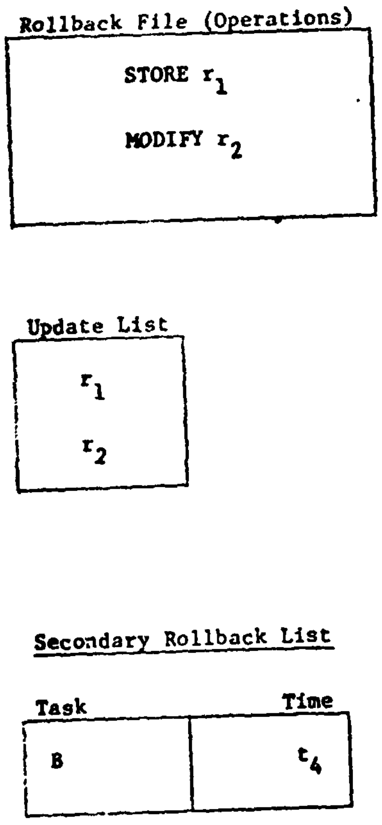
4. The log file is read backwards in order to locate, for each task in the secondary rollback list, the restart point immediately preceding the time at which the incorrect operation was detected.

5. Messages are transmitted to the host processors for the tasks in the secondary rollback list, indicating that the tasks should be rolled back to the specified restart point. The host processors will then suspend those application tasks and send the appropriate rollback information to the back-end processors for those tasks.

Figure 5 illustrates the resulting rollback file, update list, and secondary rollback list if the sample application task A of Figure 4 were to be rolled back. The potential shared data list of Figure 2 is assumed in this example.

Careful observation of Figures 2, 4, and 5 will lead to the following conclusions:

1. C is not rolled back since it accesses r_2 before A updates r_2.



Actions

1. Task A will be rolled back to time t_0.
2. r_1 will be removed from the data base.
3. r_2 will be restored to its status before time t_5.

Figure 5

Results of Rolling Back Task A

2. D is not rolled back since r_1 is not in the potential shared data list (i.e. it is not a critical record for D).

3. The MODIFY r_1 command is not rolled back since it is preceded by a STORE r_1 command.

4. Although it is not explicitly shown in the figures, the GET r_2 command at time t_{10} will be ignored during rollback processing, since B will have been added to the secondary rollback list after processing the log file entry for time t_4.

3.4 Responsibilities of Data Base Administrator

The distribution of a DBMS over a computer network enormously complicates the data base administration function. If a recovery scheme similar to that proposed in this paper is implemented, the DBA must make decisions that will have substantial impact on the time required for recovery. Perhaps the most important factors are the distribution of the data across machines and the amount of data integration. As distribution and integration of data increase, the data base becomes more accessible to a larger number of users. At the same time, recovery operations become increasingly complex. This quandry reduces to the classical data processing tradeoff between flexibility and efficiency. It is the role of the data base administrator to balance these two seemingly conflicting factors.

4. Conclusion

The rollback of an application task by a back-end processor is the central element in any recovery scheme for a distributed data base management system. In a highly integrated data base, the procedure may have to be repeated several times until no new secondary rollback lists are generated. Rollback can be executed concurrently on distinct back-end processors. However, a back-end processor can rollback only one application task at a time. It is important to note that the recovery procedure does not allow for redundant copies of data at different back-end processors. Although redundancy is unnecessary in a truly distributed DBMS, there are certain circumstances in which practical considerations may justify redundancy [1,3]. As in all other situations of this nature, the DBA must decide on redundancy.

The rollback procedure presented in this paper provides a mechanism for efficient and complete recovery in a distributed data base management system. The development of an efficient recovery mechanism can have a significant impact on the design and usage of distributed data base systems. One important design aspect that recovery can impact is deadlock handling. Due to potential inefficiencies in recovery of distributed data bases, it has been argued that deadlock prevention is more efficient than deadlock detection for a distributed DBMS [11]. However, an efficient recovery mechanism can make deadlock detection more attractive.

An efficient and complete recovery technique could provide data base users with sufficient confidence in a DBMS to allow the formulation of integrated, distributed data bases. Without that confidence, the progress in distributed data base systems will be severely impeded by lack of acceptance in the data processing environment. The recovery procedure presented in this report is a step toward developing that user confidence in data base systems.

Bibliography

1. Aschim, F., "Data Base Networks--An Overview," Management Informatics, Vol. 3,1, Feb. 1974, pp. 12-28.

2. Baum, R. I. and Hsiao, D. K., "Database Computers--A Step Towards Data Utilities," IEEE Trans. on Computers, Vol. C-25, No. 12, Dec. 1976, pp. 1254-1259.

3. Booth, G. M., "The Use of Distributed Data Bases in Information Networks, Proc 1st International Conference and Computer Communication: Impacts and Implications, Oct. 1972, pp. 371-376.

4. Booth, G. M. "Distributed Information Systems," Proc AFIPS National Computer Conference, Vol. 45, June 1976, pp. 789-794.

5. Canaday, R. E. et al., "A Back-End Computer for Data Base Management," CACM, Vol. 17, No. 10, Oct. 1974, pp.575-582.

6. Lowenthal, E. I., "The Back-End Computer," MRI Systems Corp., P.O. Box 9968, Austin, Texas 78766, Apr. 1976.

7. Maryanski, F. J., et al., "A Minicomputer Based Distributed Data Base Management System," Proc NBS-IEEE Trends and Applications Symposiem: Micro and Mini Systems, May 1976, pp.113-117.

8. CODASYL Data Description Language Journal of Development, Document C1362: 113, U.S. Government Printing Office, Washington, D.C., 1973.

9. Fossum, B. M., "Data Base Integrity as Provided for by a Particular Data Base Management System," in Data Base Management, Klimbie, J. W. and Koffeman, K. L. (eds.), North Holland, Apr. 1974, pp. 271-288.

10. Maryanski, F. J., Fisher, P. S., and Wallentine, V. E., "A User-Transparent Mechanism for the Distribution of a CODASYL Data Base Management System," TR CS 76-22, Computer Science Department, Kansas State University, Manhattan, Kansas 66506, Dec. 1976.

11. Chu, W. W., and Ohlmacher, G. "Avoiding Deadlocks in Distributed Data Bases," Proc ACM Annual Conference, Nov. 1974, pp. 156-160.

FILME
9